



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**CREATING AN AUTONOMOUS MULTI-UAV
COMMUNICATIONS RELAY NETWORK**

by

James R. McMasters

June 2017

Thesis Advisor:

Second Reader:

Weilian Su

Tri T. Ha

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE June 2017	3. REPORT TYPE AND DATES COVERED Master's Thesis 06-27-2016 to 06-16-2017		
4. TITLE AND SUBTITLE CREATING AN AUTONOMOUS MULTI-UAV COMMUNICATIONS RELAY NETWORK		5. FUNDING NUMBERS		
6. AUTHOR(S) James R. McMasters				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943		8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		10. SPONSORING / MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.		12b. DISTRIBUTION CODE		
13. ABSTRACT (maximum 200 words) Unmanned aerial vehicles (UAVs) are used in a wide variety of applications. In the military, these systems can act as a communication relay between ground units and offshore units or enhance surveillance and reconnaissance; however, the low transmission power of common wireless networking equipment and obstacles limit the range of these systems. A multi-UAV relay network addresses this problem, providing increased range and coverage. The main contribution of this thesis is the design and implementation of a scalable UAV network that makes use of inexpensive IEEE 802.11 routers and Raspberry Pis to control six commercial quadcopter UAVs simultaneously. A platform-agnostic, GUI program for multi-UAV monitoring and control was implemented, and additional programs for networking, communication, and data transfer were written. The resulting network was tested indoors, and its performance was analyzed.				
14. SUBJECT TERMS UAV, communications relay, wireless network			15. NUMBER OF PAGES 73	16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**CREATING AN AUTONOMOUS MULTI-UAV COMMUNICATIONS RELAY
NETWORK**

James R. McMasters
Ensign, United States Navy
B.S., United States Naval Academy, 2016

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
June 2017**

Approved by: Weilian Su
Thesis Advisor

Tri T. Ha
Second Reader

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Unmanned aerial vehicles (UAVs) are used in a wide variety of applications. In the military, these systems can act as a communication relay between ground units and offshore units or enhance surveillance and reconnaissance; however, the low transmission power of common wireless networking equipment and obstacles limit the range of these systems. A multi-UAV relay network addresses this problem, providing increased range and coverage. The main contribution of this thesis is the design and implementation of a scalable UAV network that makes use of inexpensive IEEE 802.11 routers and Raspberry Pis to control six commercial quadcopter UAVs simultaneously. A platform-agnostic, GUI program for multi-UAV monitoring and control was implemented, and additional programs for networking, communication, and data transfer were written. The resulting network was tested indoors, and its performance was analyzed.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1	Introduction	1
1.1	Background	1
1.2	Objectives	2
1.3	Thesis Organization	4
2	Quadcopter UAV Technology	7
2.1	Quadcopter Choice	7
2.2	DJI Matrice 100 Quadcopter Description	8
3	Network Design and Programming	13
3.1	Network Design Considerations	13
3.2	Network Topology	15
3.3	Network Architecture	15
3.4	Routing Architecture	16
3.5	Network Programming	17
4	UAV Assembly and Network Configuration	21
4.1	Matrice 100 Assembly	21
4.2	Initial Configuration of Components	22
4.3	Powering the Payload Devices	25
4.4	Mounting the Payload	26
5	Control and Communications Programming	27
5.1	Flight Controller Program	27
5.2	GUI Control Program	34
5.3	Communications Program	39
6	Simulation, Testing, and Performance Evaluation	43

6.1	Simulation Test Flights	43
6.2	Network Performance Evaluation	43
7	Conclusions and Recommendations	49
7.1	Summary of Work	49
7.2	Future Work	49
	List of References	51
	Initial Distribution List	55

List of Figures

Figure 1.1	A Generic UAV System. Source: [2].	2
Figure 2.1	The DJI Matrice 100 Platform. Source: [12].	8
Figure 2.2	The DJI Guidance System. Source: [14].	11
Figure 2.3	The Generic Hardware Setup for DJI’s Onboard SDK. Source: [15].	12
Figure 3.1	Chain (top) and Cyclic (bottom) Topologies for Information Trans- fer. Adapted from [18].	16
Figure 3.2	Multi-UAV Network Architecture	17
Figure 4.1	The Matrice 100’s Reserved Ports. Source: [20].	25
Figure 4.2	Converter Circuit Used to Power the Payload on a Matrice 100 . .	26
Figure 4.3	The Payload Mounted on a Matrice 100	26
Figure 5.1	The DJI Onboard SDK Programming Workflow. Source: [28]. . .	28
Figure 5.2	The Onboard SDK Sample Program. Source: [29].	29
Figure 5.3	Spectrum Analyzer Trace with Linksys WRT54GL Turned On . .	30
Figure 5.4	Spectrum Analyzer Trace with Matrice 100 Remote Controller Turned On	31
Figure 5.5	High-Level Flow of Control of the Flight Controller Program . .	32
Figure 5.6	The “Lawnmower Up” (left) and “Perimeter Up” (right) Flight Pat- terns	34
Figure 5.7	High-Level Flow of Control of the GUI Control Program	35
Figure 5.8	The GUI for Controlling a Network of Six UAVs	36
Figure 5.9	Example Usage of VoIP Service	40

Figure 5.10	The GUI for Creating a VoIP Session	40
Figure 6.1	Successful Test of the GUI’s “Show Map” Feature	44
Figure 6.2	WinSCP 50-MB File Transfer Results	45
Figure 6.3	iperf Multi-Hop TCP Throughput Results	46

List of Tables

Table 2.1	Basic DJI Matrice 100 Specifications. Adapted from [13].	10
Table 3.1	Basic Linksys WRT54GL Specifications [16]	14
Table 3.2	Basic Raspberry Pi 3 Model B Specifications. Adapted from [17].	15
Table 3.3	IP Address Assignments for the Wireless Routers	18
Table 3.4	IP Address Assignments for All Network Devices	19
Table 4.1	SSH Tunnel Routes for Connecting to a Particular UAV	24

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

API	Application Programming Interface
CAN	Controller Area Network
COTS	commercial off-the-shelf
CSV	comma-separated values
DHCP	Dynamic Host Configuration Protocol
DJI	Dà-Jiāng Innovations Science and Technology Co., Ltd
GCS	ground control station
GPS	Global Positioning System
GUI	graphical user interface
IP	Internet Protocol
ISM	industrial, scientific, and medical radio
LOS	line-of-sight
LPI	low probability of intercept
M100	Matrice 100
MANET	mobile ad hoc network
RC	remote controller
RTH	return-to-home
SDK	Software Development Kit

SSH	secure shell
SSID	service set identifier
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UAV	unmanned aerial vehicle
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol (IP)

Acknowledgments

I would like to thank Dr. Weilian Su for his guidance on this project. I would also like to thank Jeff Knight for his support and assistance in designing and assembling the power circuits for this project. He was extremely generous with his time and resources.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:

Introduction

Communication between ground units and offshore units is essential in today's military operations. Conventionally, satellites have been used as communication relays to establish this link. The satellites are effective but leave room for improvement in cost, speed, and coverage. As such, the Department of Defense desires a faster and less expensive solution that is rapidly deployable and adaptive to changing operational conditions [1]. Formations of unmanned aerial vehicles (UAVs) that act as intermediaries between ground units, satellite relays, and offshore units can address this problem and improve upon the satellite solution.

1.1 Background

A UAV is an aircraft that can be operated autonomously or remotely by a pilot. The setup for a typical UAV system is shown in Figure 1.1. The ground control station consists of the antennas, computers, and other equipment required to operate the UAV, such as a remote controller (RC). In general, the ground control station (GCS) must be within the line-of-sight (LOS) of the UAV in order to transmit control signals and receive information from the UAV. If the UAV is flown beyond the LOS of the GCS, then a satellite may be used as a relay to control it. In place of a relay satellite, an intermediary UAV may also serve as a relay for control. If multiple UAVs are chained together in this manner, a communications relay network can be created.

The creation of a formation of UAVs poses a significant design challenge. The wireless communication architecture, algorithms for coordinated control, and need for obstacle avoidance are only a few of the factors that must be considered. Nonetheless, the use of one or more UAV nodes as communication relays is a more attractive option than the satellite method under certain circumstances. First, satellites are much more expensive than UAVs and require more time to deploy. Second, UAVs are highly maneuverable. For instance, in obstacle-laden environments, UAVs can be positioned perfectly to provide coverage in areas where satellites cannot. Furthermore, directional antennas may be used

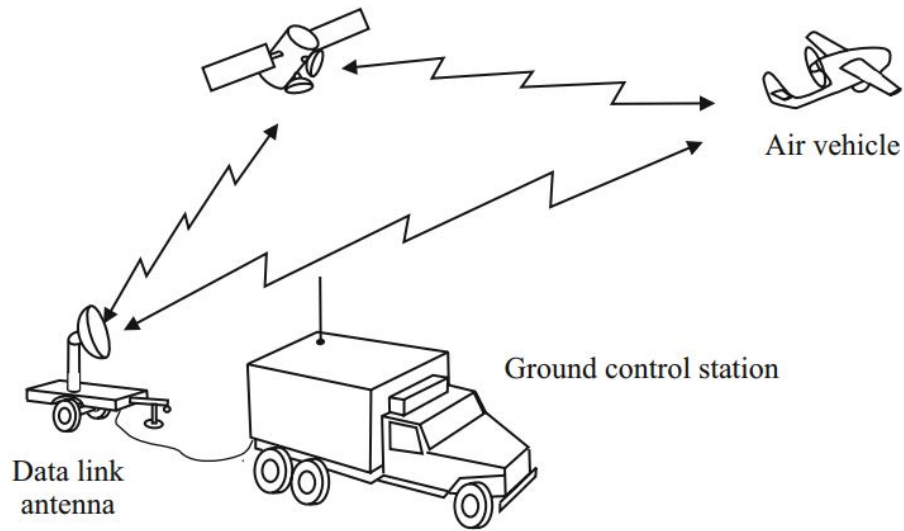


Figure 1.1. A Generic UAV System. Source: [2].

to pinpoint the coverage area and reduce the probability of intercept, providing more secure communications than satellites offer.

The main drawbacks of UAV technology are limited battery life and inability to fly in poor weather conditions. Despite these faults, UAVs may still be a viable solution for the purpose of forming short-term, range-extending networks. As long as multiple UAVs can be formed into a secure, reliable network with sufficient throughput and bandwidth to support all of the expected users and traffic, then a communication link between ground units and offshore ships can be realized.

1.2 Objectives

The main purpose of this thesis is to design and build a wireless network of six commercial off-the-shelf (COTS) UAVs capable of transferring user data and flight control information. At the very least, this requires the following components:

- Choose appropriate UAV platforms and networking equipment.
- Design the network architecture.
- Assemble and configure the UAVs and networking equipment according to the network design.

- Implement scalable control software for the formation of UAVs.
- Implement software for setup, communication, and teardown of the network.
- Perform flight tests to validate the control software.
- Benchmark the network performance.

1.2.1 Related Work

In [3], Yanar investigated the use of directional antennas on UAVs, determined the best flight paths for network coverage, and simulated various medium access and routing protocols in UAVs networks to determine which ones performed the best. He concluded that directional antennas can provide advantages in signals intelligence and greater range and reliability of communications. The best configuration and flight patterns for UAVs in this network was determined to be a pair of UAVs flying in a double-lawnmower pattern at an altitude of 1 km using IEEE 802.11 MAC protocols and DYMO routing.

Likewise, an aerial mobile ad hoc network (MANET) was designed and implemented in [4]. The researchers made use of the IEEE 802.11s standard for wireless mesh networks. The main contributions of their work was an extensible architecture for wirelessly networking multiple UAVs and platform-agnostic software for flight control and autonomous network creation. Their network was ideal for emergency response in high mobility situations, which drove their decision for making the network topology fully-meshed.

A two-UAV communication relay solution was developed in [5]. The test platforms were quadcopters with Pixhawk flight controllers and Raspberry Pi 2 Model B computers as additional payload. Indoor and outdoor tests were carried out to ascertain the data rate and maximum range extension that was possible. The researchers were focused on maintaining good quality communications for control signals and maximizing range, so they used 915-MHz radios, which gave ranges up to a few kilometers but much lower data rates. Autonomy was not explored in this study.

A similar project involving two fixed-wing, relay UAVs was undertaken in [6]. This study stressed the importance of field experimentation using COTS wireless equipment. They conducted experiments for UAV networks with COTS 2.4 GHz and 5.0-GHz 802.11 equipment. They performed several experiments to determine the effect of changing antenna orientation and placement and demonstrated network throughputs exceeding 1 Mbit/s.

In [7], 5-GHz 802.11n USB dongles were installed on a pair of fixed-wing UAVs. They found that proper antenna orientation had a significant impact on throughput. Although they observed User Datagram Protocol (UDP) throughputs of up to 176 Mbit/s in a laboratory environment, the throughputs in their aerial experiments only reached 19 Mbit/s at short distances. They concluded that the 802.11n chipsets in their USB dongles were unable to handle the high mobility of the UAVs. Similar conclusions about the negative effect of mobility and the need for proper antenna orientation were reached in [8], in which the researchers' main contribution was the design of an antenna extension specifically constructed to enhance the performance of UAV networks.

Comparable studies involving two UAVs were conducted in [9] and [10]. The main findings in [9] were that one-hop and two-hop communications could be handled by intermediary UAVs acting as access points or relaying mesh points, but multi-hop UAV networks require new network architectures. They concluded that the current state of the 802.11s standard was not optimized for networking UAVs in that it did not take into account the unique parameters involved in networking highly mobile, aerial nodes. Also, the researchers made an important realization that most UAV remote controller's operate in the 2.4-GHz band and decided to use 802.11a, which operates at 5.2 GHz, to avoid interference. In [10], the researchers setup an ad hoc connection between two UAVs equipped with 802.11b/g routers and directional antennas and tested the end-to-end throughput of the link under various conditions. They concluded that with high UAV mobility and increased distance between UAVs, the performance of the link was degraded.

Finally, algorithms for determining the optimal position for aerial relay nodes in a multi-UAV communications network were developed in [11].

Notably, none of these studies makes use of more than one relaying UAV, so their network designs do not face the unique challenge of determining the best way to interconnect a large number of nodes. This problem is a key aspect of this thesis' network design.

1.3 Thesis Organization

This thesis has seven chapters. In Chapter 2, quadcopter UAV technology and the advantages and disadvantages of the specific platform purchased for this project are discussed. The

network design is discussed in Chapter 3, and the assembly and configuration of all of the project's components is described in Chapter 4. The details of the software written for controlling all of the UAVs are discussed in Chapter 5. Simulations and network performance analysis are presented in Chapter 6. Finally, conclusions and future work recommendations are provided in Chapter 7.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:

Quadcopter UAV Technology

There are many different types of UAVs, including fixed-wing, rotary-wing, and multi-rotor. In this chapter, the advantages and disadvantages of quadcopter UAVs are discussed. Additionally, the platform purchased for this thesis is introduced.

2.1 Quadcopter Choice

For this thesis research, we required UAVs capable of acting as semi-stationary nodes, adaptable to harsh terrain, and rapidly deployable in large numbers in a communications relay network. Some viable options included fixed-wing, multirotor, and helium-filled balloons. Quadcopter UAVs were chosen because of their versatility. Specifically, quadcopters offer superior maneuverability, portability, price, size, and mechanical simplicity.

In terms of maneuverability, quadcopter systems are more capable than others due to their ability to take off and land vertically, hover, and navigate in small areas. These characteristics are especially useful in rough terrain or dense urban areas where mountains or man-made obstacles can impede their flight paths. In these environments, small quadcopters can be controlled with enough precision to avoid obstacles and fit into tight spaces. Furthermore, quadcopter systems are usually lightweight and easy to store, whereas fixed-wing UAVs and other types are larger and often require specialized launch and retrieval equipment to be stored.

Additionally, the chosen platform must be able to carry any networking equipment and sensors necessary for creating the network. Fixed-wing UAVs certainly meet this requirement, but most quadcopters are significantly more constrained in this area. Fortunately, the continual miniaturization of electronics has made room for small quadcopter UAVs to increase their payload enough to meet this requirement. Other types of UAVs address some but not all of the desired criteria, leaving quadcopters as the best option.

2.2 DJI Matrice 100 Quadcopter Description

The quadcopter platform considered for this study is shown in Figure 2.1. The Matrice 100 (M100) was developed by Dà-Jiāng Innovations Science and Technology Co., Ltd (DJI) as a developer platform for business professionals and researchers. As such, DJI opened their Software Development Kit (SDK) to enable developers to create their own applications. Several characteristics of the M100 are listed in Table 2.1. Most notably, the platform can carry a payload of up to 1 kg, and its lightweight frame and ability to carry two batteries give it a maximum flight time of 40 minutes when hovering with no additional payload.



Figure 2.1. The DJI Matrice 100 Platform. Source: [12].

In addition to the features in Table 2.1, several more of the platform's properties are summarized below:

Robust Flight Controller - DJI's flight controller maintains the aircraft's stability, allowing the developer to focus all of their attention on their specific application,

Customizable Mounting and Additional Expansion Bays - additional compartments for batteries and payload devices are provided and can be attached to the airframe wherever is best for balance,

Hovering Accuracy - the default sensors offer 50 cm vertical accuracy and 2.5 m horizontal accuracy,

Reserved Serial Ports - two parallel Controller Area Network (CAN) ports and two parallel Universal Asynchronous Receiver/Transmitter (UART) ports for serial communication with the flight controller,

Reserved Power Ports - XT60 and EC3 ports for powering payload devices,

Flight Modes - waypoint mission, point of interest loitering, course lock, home lock, and custom (application-specific) modes are available.

Aside from these characteristics, most information about the electronic subsystems onboard the M100 is not disclosed by DJI, likely because the rest of their consumer-grade platforms use the same technologies.

The main drawbacks of the M100 are its payload limit, battery life, and the proprietary nature of some of its subsystems. With a 1-kg payload, the Matrice 100 is expected to have a maximum flight time of approximately 30 minutes. Additionally, the batteries will power the onboard embedded systems for this project, which further reduces the flight time. This limits the types of operations for which the network will be useful and makes testing cumbersome. With this in mind, thirty batteries were purchased, which allows testing to continue without pausing while batteries charge.

In addition, the protocol (DJI Lightbridge) used for transmitting video and control signals between the aircraft and the RC is proprietary. DJI states that Lightbridge operates at 2.4 GHz and has a maximum communication range of 5 km; however, without a better understanding of how Lightbridge works, the security of the protocol cannot be validated. As a result, we do not rely on Lightbridge for this project and instead create our own separate links for control and communications.

2.2.1 DJI Matrice 100 Ground Station

The default GCS for the M100 consists of DJI's C1 remote controller with a smartphone or tablet connected via USB cable. The RC uses DJI's Lightbridge protocol to communicate with the Matrice 100's N1 flight controller and operates in the 2.4-GHz band with a maximum range of 5 km [13]. With this station, the pilot may send commands to the platform and receive telemetry data from the flight controller. This data, including Global Positioning System (GPS) location and video, is displayed on the smartphone or tablet in the DJI Go application. On the other hand, the dedicated RC is used for fine-grain flight

Table 2.1. Basic DJI Matrice 100 Specifications. Adapted from [13].

Parameter	Value
Diagonal Wheelbase	650 mm
Weight	2355 g (with TB47D battery)
Max. Payload	1000 g
Max. Takeoff Weight	3600 g
Hovering Time	40 min (with two TB48D batteries)
Max. Speed (ATTI mode)	22 m/s (no payload, no wind)
Max. Speed (GPS mode)	17 m/s (no payload, no wind)

control of the aircraft. Different types of ground stations can be implemented with the onboard SDK, as described in Chapter 5. In this case, the C1 remote controller is used as a secondary, fail-safe mechanism.

2.2.2 Guidance System

A high level of accuracy is required to achieve autonomous flight control of multiple UAVs if they are operating in close proximity or the environment has obstacles. Consequently, each M100 platform was equipped with DJI's Guidance system. The Guidance system is an accessory that gives the M100 advanced visual tracking and obstacle avoidance. The system is a sensor-based navigation aid that uses ultrasonic sensors and cameras to provide real-time information about its surroundings. This data, including velocity, position, and obstacle clearance, is sent to the RC. Given sufficient lighting and faced with texture-rich surfaces, the system can provide a velocity measurement accuracy of 0.04 m/s and position accuracy of 50.0 cm at 2.0 m above the ground. This allows visual positioning in GPS-denied areas and obstacle detection and avoidance, which are crucial for autonomous flight. The sensors are functional in the range from 20.0 cm to 20.0 m [14]. The system is shown in Figure 2.2. Of note, the system consists of five components, so either the top or bottom face of the aircraft will not have a sensor, depending on how the system is installed.

Without further configuration, Guidance provides obstacle detection and prevention of collision. If an operator attempts to fly a Matrice 100 equipped with Guidance toward a wall, the Guidance system detects the wall and overrides the user's command in order to prevent a collision. The distance at which the Guidance system stops the quadcopter is user-



Figure 2.2. The DJI Guidance System. Source: [14].

configurable. With additional programming that leverages the Guidance SDK, autonomous obstacle avoidance can be achieved. For instance, an algorithm for navigating around obstacles that makes use of the each of the sensors' outputs could be implemented.

2.2.3 The DJI Software Development Kits

One key feature of the Matrice 100 is that its behavior can be programmed with DJI's various SDKs. There are three SDKs: mobile, onboard, and guidance. The mobile SDK is used for creating mobile phone applications. Examples include three-dimensional mapping and autopilot applications. The onboard SDK is for creating applications that run on a small embedded system that the quadcopter carries. This device may be connected via UART cables to the Matrice 100's flight controller to control flight, gather flight telemetry data, transmit data, control the quadcopters gimbal and camera (if attached), and provide advanced sensing and flight algorithms. The programmer can use the provided C/C++ source APIs to send and receive data over the serial port or make use of DJI's protocol to write their own communication drivers. The generic setup for using the onboard SDK is shown in Figure 2.3.

For this project, the onboard SDK is more suitable than the mobile SDK for several reasons. First, if the mobile SDK is used, the programmer must rely on DJI's remote controller as its transceiver. DJI does not provide a recommended way to use COTS RCs with their platforms, which forces the developer to rely on DJI's proprietary controller channel. On



Figure 2.3. The Generic Hardware Setup for DJI's Onboard SDK.
Source: [15].

the other hand, if the onboard SDK is used, the developer is free to implement a way to send signals to the onboard device, which, in turn, is directly connected to the UAV and uses DJI's serial communications protocol to transfer data to the flight controller. Additionally, if the mobile SDK is used, the complexity of the applications that can be created is limited by the computing power of the mobile device. For 3D mapping, aerial imagery, and other applications that are not time critical, the mobile SDK is suitable and easier to use; however, for the development of a secure, multi-UAV network, the onboard SDK is the better option.

CHAPTER 3:

Network Design and Programming

Before the UAVs were assembled and the control programs were written, the network was designed. In this chapter, the basic requirements for creating a wireless network of mobile UAVs are outlined, and the design decisions that were made are discussed. The final network architecture and the means by which data is transferred throughout the network is explained.

3.1 Network Design Considerations

Any network of UAVs will likely have the following features in common: wireless connectivity, mobile nodes, and a partial or fully-mesh topology. Furthermore, for this thesis' particular network, all of the following are important design considerations:

- Weight** - payload devices must not exceed the aircraft's payload limit to ensure flight stability and maximize flight time;
- Power consumption** - payload devices should be low power so that the bulk of the battery is reserved for the aircraft's propulsion system;
- Security** - the network should be secure from intruders hijacking or eavesdropping, among other exploits;
- Scalability** - the network should be able to scale to support the expected number of users and provide reasonable data rates.

In addition, the wireless networking equipment should be able to be modified to carry directional antennas in order to provide low probability of intercept (LPI) and better signal strength to particular locations. With these considerations in mind, the most appropriate wireless networking equipment was determined. Several options exist, including wireless routers, repeaters, and lightweight mesh-networking equipment. The main advantages of mesh networks are their self-forming ability and resilience; however, their throughput takes a hit due to the management and rebroadcasting that must take place to maintain the network, which also causes their power consumption to be quite high.

Wireless repeaters are able to extend the range of a network by receiving, amplifying, and rebroadcasting an incoming wireless signal. Most COTS wireless repeaters that were examined were either too heavy or not suitable for other reasons. For one, in terms of scalability, wireless repeaters are a poor choice because instead of creating new subnetworks they simply retransmit a parent network's signal; thus, the final network architecture would be a single network, which is far from ideal.

Clearly, although repeaters and mesh devices offer some of the desired behavior for creating a wireless communications relay network, a brief survey of their limitations shows that they are lacking in many areas. Wireless routers, on the other hand, possess all the desired features. First, there are a large number of reliable, inexpensive, lightweight COTS wireless routers to choose from. They are low power devices and many can be modified to act as mesh access points and repeaters as well. The router that was chosen for this thesis was the Linksys WRT54GL. Its pertinent features are listed in Table 3.1.

The next step was to determine the best way to interconnect the wireless nodes in the network. To avoid the pitfalls of wireless mesh networking and repeaters, multi-hop secure shell (SSH) tunneling through Raspberry Pis was used. Some relevant specifications of the Raspberry Pi Model 3 B are listed in Table 3.2. Raspberry Pis are ideal payload devices because they are lightweight, powerful, and versatile. These microcomputers were chosen not only because of their ability to interconnect the network but also because they are compatible with the Matrice 100's onboard SDK and can be used for flight control.

Table 3.1. Basic Linksys WRT54GL Specifications [16]

Parameter	Value
Weight	482 g
Power Requirements	12 VDC, 0.5 A
IEEE WiFi Standard	802.11b, 802.11g
Ports	Four 10/100 RJ-45 Ethernet
Firmware	Customizable (tomato and DD-WRT compatible)
Security	WPA/WPA2 Encryption
Antennas	Omnidirectional (replaceable)

Table 3.2. Basic Raspberry Pi 3 Model B Specifications. Adapted from [17].

Parameter	Value
CPU	1.2 GHz 64-bit quad-core ARMv8
RAM	1GB SDRAM
Storage	microSD
Power Requirements	5 VDC, 1.2 A
Ethernet Port	10/100 RJ-45
WiFi	built-in 802.11n
USB Ports	4
Weight	42 g
Dimensions	3.35" $\times$ 2.2" $\times$ 0.8"

3.2 Network Topology

The network design must take into account the topology for information exchange. Network topology refers to the physical or logical arrangement of a network's nodes and links. Two basic topologies are shown in Figure 3.1. In a cyclic topology, each aircraft exchanges two-way information with its neighbors, whereas in a chain topology, there is no loop. Another option is a fully-meshed topology, where all aircraft exchange information with every other aircraft in the formation; however, mesh topologies are impractical due to the high delays that would be experienced in such a formation. In practice, UAV formations will use some combination of cyclic and chain topologies. In this thesis, a chain topology was implemented. In this arrangement, adjacent nodes must be within each other's range of coverage, and nodes that are outside of one another's transmission range can reach each other through the intermediary nodes between them.

3.3 Network Architecture

The overarching network architecture that this thesis fits into has several layers. The uppermost layer has satellites, and the middle layers consist of UAVs at varying altitudes. In this thesis research, the lowest two layers are the UAVs and the ground users and controllers that connect to and transfer data in the network. The network architecture for these lower two layers is shown in Figure 3.2. Each quadcopter is equipped with four Raspberry Pi 3 Model B computers and one Linksys WRT54GL router. The GCS consists of a laptop and

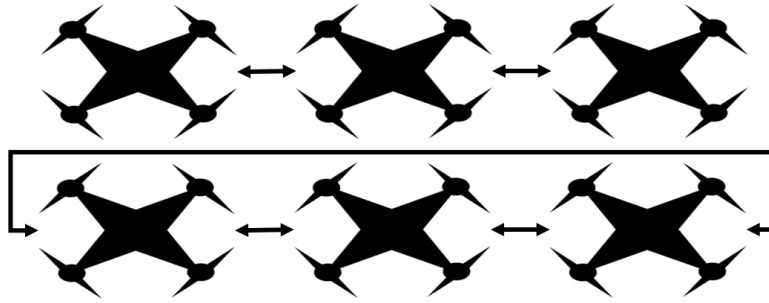


Figure 3.1. Chain (top) and Cyclic (bottom) Topologies for Information Transfer. Adapted from [18].

a router, and one of the Raspberry Pis on each quadcopter is associated with the GCS's access point.

Each quadcopter belongs to multiple different subnets. A subnet is a logical division of a network. In this case, the potential subnets are as follows:

1. A pair of quadcopters within the same architectural layer
2. A pair of quadcopters in different layers (such as lower-to-upper)
3. A quadcopter paired with a ground layer interface

These subnets are formed by establishing wireless connections between the Raspberry Pis and the routers. One of the Raspberry Pis is always reserved for connecting to the GCS if they are within range of one another. This Raspberry Pi is also reserved for serially connecting to the flight controller of the quadcopter on which it is equipped. The remaining Raspberry Pis are capable of connecting to the routers of any of the other quadcopters. For example, in Figure 3.2, the lowermost Raspberry Pi on “drone0” is connected to the ground router via Wi-Fi. Likewise, “drone1” has a wireless connection to both “drone0” and “drone1,” as indicated by the blue Wi-Fi symbols radiating toward the corresponding subnets.

3.4 Routing Architecture

In this network architecture, the routers do not perform network routing. Instead, transparent multi-hop SSH is used to establish connections throughout the network. SSH is an

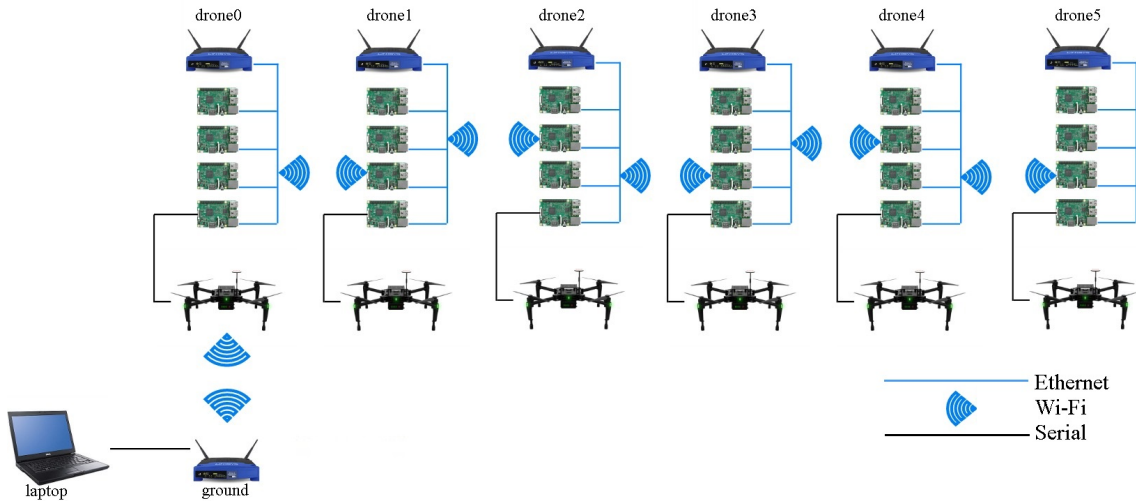


Figure 3.2. Multi-UAV Network Architecture

application layer protocol primarily used for creating a secure connection between two computers. SSH offers both password authentication and RSA authentication for establishing a connection, and AES and HMAC-MD5 are used for confidentiality and integrity, respectively [19]. When SSH is used to create a multi-hop connection between more than two computers, it is referred to as multi-hop SSH. For example, if a command needs to be sent to the third quadcopter in the network, an SSH tunnel is created from the GCS to the third quadcopter and then the desired command is executed. In implementation, this requires careful key distribution between the Raspberry Pis that are used for tunneling. In addition, an Internet Protocol (IP) addressing scheme that is scalable and allows for automation must be created.

3.5 Network Programming

Two network-related programs were written for this project: the flight control program and the program that sets up the communication tunnel for voice and text communications and data transfer. Both of these programs rely on the network being configured with static IP addresses according to a specific scheme. The IP addressing scheme for the current network is shown in both Table 3.3 and Table 3.4.

First, the Raspberry Pis were named according to the UAV to which they were mounted. For example, the four Raspberry Pis with names starting with “pi0” were mounted to “drone0.” Accordingly, each of these Raspberry Pis was connected via Ethernet to the router on “drone0,” so they all belong to the 192.168.0.1 subnet. Furthermore, the Raspberry Pis named “piX_0,” where $X = 0, 1, 2, \dots, 5$, correspond to the lowermost Raspberry Pis on each UAV in Figure 3.2, which explains why all of their wireless interfaces were assigned IP addresses within the “ground” router’s subnet. These Raspberry Pis are called the UAV controller Raspberry Pis for the remainder of this thesis. The remaining Raspberry Pis were assigned IP addresses such that the peer-to-peer connections shown in Figure 3.2 were realized. For instance, the wireless interfaces of “pi1_1” and “pi1_2” were assigned static IP addresses of 192.168.0.6 and 192.168.2.6, respectively; thus, these two Raspberry Pis create connections from the “drone1” subnet to the “drone0” and “drone2” subnets, respectively. Finally, the Raspberry Pis with unassigned wireless interfaces could connect to any additional access points that are added to the network.

Each of the programs that was made for this project kept this IP addressing scheme in mind and is capable of scaling to handle the addition of new UAVs and other nodes that follow this scheme.

Table 3.3. IP Address Assignments for the Wireless Routers

SSID	IP Address
ground	192.168.255.1
drone0	192.168.0.1
drone1	192.168.1.1
drone2	192.168.2.1
drone3	192.168.3.1
drone4	192.168.4.1
drone5	192.168.5.1

Table 3.4. IP Address Assignments for All Network Devices

Device	Wired Interface	Wireless Interface
Control Laptop	unassigned	192.168.255.2
pi0_0	192.168.0.2	192.168.255.3
pi0_1	192.168.0.3	192.168.1.6
pi0_2	192.168.0.4	unassigned
pi0_3	192.168.0.5	unassigned
pi1_0	192.168.1.2	192.168.255.4
pi1_1	192.168.1.3	192.168.0.6
pi1_2	192.168.1.4	192.168.2.6
pi1_3	192.168.1.5	unassigned
pi2_0	192.168.2.2	192.168.255.5
pi2_1	192.168.2.3	192.168.1.7
pi2_2	192.168.2.4	192.168.3.6
pi2_3	192.168.2.5	unassigned
pi3_0	192.168.3.2	192.168.255.6
pi3_1	192.168.3.3	192.168.2.7
pi3_2	192.168.3.4	192.168.4.6
pi3_3	192.168.3.5	unassigned
pi4_0	192.168.4.2	192.168.255.7
pi4_1	192.168.4.3	192.168.3.7
pi4_2	192.168.4.4	192.168.5.6
pi4_3	192.168.4.5	unassigned
pi5_0	192.168.5.2	192.168.255.8
pi5_1	192.168.5.3	192.168.4.7
pi5_2	192.168.5.4	unassigned
pi5_3	192.168.5.5	unassigned

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:

UAV Assembly and Network Configuration

Each component in the network required some form of assembly and configuration to function properly, including the quadcopters, Raspberry Pis, routers, and the various cables and power circuits for the payload devices. A brief discussion of this configuration process is provided in this chapter.

4.1 Matrice 100 Assembly

The M100 quadcopters required assembly upon arrival. The official M100 user manual and assembly videos were used as guides during the assembly process [20]–[23]. A few parts of the assembly were ambiguous or provided choices for the user to make based on his or her preferences; thus, a brief description of the assembly of the UAVs is provided in this section to explain the decisions that were made.

The following assembly steps were applied to each of the six M100s. First, the frame arms were mounted to the center frame. The kit came with three different fasteners: curved, 0°, and 3°. The 3° fasteners were used because the user manual recommended them for better flight performance. Additionally, propeller guards were fastened to the outside of each frame arm.

Next, the battery compartments were mounted to the center frame. One was mounted on top, and the other was mounted on bottom. Two expansion bay sets came with each M100. Six additional expansion bay sets were purchased for mounting future devices. Each of these was assembled according to the user manual. Two were mounted to the top of the center frame, and one was mounted to the bottom. The bottom expansion bay was reserved for mounting the Guidance system.

All five sensors of the Guidance system were mounted to the center frame on the bottom expansion bay. Next, the GPS module was mounted. There were mounting positions on either the left or right side of the center frame for the GPS module. For each M100, the GPS module was mounted to the right-side mounting position.

Finally, although no gimbals or cameras were purchased for this project, the supplied gimbal mounting plates were installed on each aircraft.

4.2 Initial Configuration of Components

The next step was to configure each component to meet the requirements of the network design.

4.2.1 Matrice 100 Configuration

The flight controllers, batteries, Guidance systems, and remote controllers all have firmware that must be updated regularly. The flight controller firmware was updated via DJI's PC Assistant 2 software. Likewise, the Guidance firmware was updated within its own configuration software. The remote controller and battery firmware were both updated within the DJI GO mobile application.

Next, the Raspberry Pis were given control over the flight controller. This was done by enabling Application Programming Interface (API) control in the DJI Assistant 2 software [24].

4.2.2 Raspberry Pi Configuration

As mentioned in Chapter 3, each M100 was equipped with four Raspberry Pis as payload; thus, 24 Raspberry Pis needed to be configured. First, the most recent version of the Raspbian operating system was installed on one of the Raspberry Pis. Raspbian is an operating system based on Debian that is optimized to run on Raspberry Pis and capable of running DJI's onboard SDK [25]. The following steps were taken after Raspbian was installed:

- Bluetooth was disabled to limit power consumption and interference,
- SSH was enabled,
- The serial interface was enabled and configured to communicate with the M100,
- The wireless interface was configured, and
- DJI's onboard SDK was installed.

The Win32DiskImager was used to clone the microSD card for this Raspberry Pi [26]. The cloned microSD image was written to the remaining 23 microSD cards. In this way, these 23 Raspberry Pis automatically received the same configuration as the manually-configured one, which greatly expedited the process of configuring all of the Raspberry Pis.

In addition, the MAC addresses of both the Ethernet and wireless interfaces of each Raspberry Pi were recorded so that the routers could assign them static IP addresses.

4.2.3 Router Configuration

The latest version of DD-WRT, a Linux-based, open source router firmware, was installed on each router [27]. This was preferred to the default firmware in that it allowed for greater customization. First, the default firmware does not provide a means for automatic wireless channel selection, whereas DD-WRT supports this feature. Without this feature, each router in the network needs to be manually configured to use non-overlapping channels. Second, DD-WRT provides a simple interface for setting up static IP addresses. Third, DD-WRT allows users to increase the router's transmission power. Finally, DD-WRT is well-documented and has an active support community. For these reasons, DD-WRT was installed on each of the Linksys WRT54GL routers in the network.

The following steps were taken to configure each router after DD-WRT was installed:

- The router's IP address and service set identifier (SSID) were set as per Table 3.3.
- The Dynamic Host Configuration Protocol (DHCP) range was defined for assigning IP addresses to hosts (except for the Raspberry Pis, which were assigned static IP addresses).
- Transmission power was set to the maximum allowed value.
- Static IP addresses were assigned to each Raspberry Pi as per Table 3.4.

4.2.4 Network Integration

Next, the Raspberry Pis and routers were connected, and the Raspberry Pis were configured to permit transparent multi-hop SSH and automated SSH tunneling through local port forwarding. First, each of the four Raspberry Pis per UAV was connected to their UAV's router with an Ethernet cable. Short Ethernet cables were made with an Ethernet kit for

this purpose. In accordance with the IP addressing scheme from Chapter 3, these four Raspberry Pis must belong to the same subnet; therefore, the routers were configured to lease static IP addresses to the Raspberry Pis based on the Raspberry Pis' MAC addresses.

Finally, the `ssh-keygen` command was used to generate public/private RSA key pairs on each Raspberry Pi. The `ssh-copy-id` command was used to share the public key of one Raspberry Pi with its peers in the network. This procedure was carried out between every Raspberry Pi involved in creating an SSH tunnel. All of the potential SSH routes for connecting to the Raspberry Pi in control of a particular UAV in the network are shown in Table 4.1. Note that the IP addresses in Table 4.1 are abbreviated to the last two octets because the first two octets are always "192.168."

Table 4.1. SSH Tunnel Routes for Connecting to a Particular UAV

Destination UAV	Route
drone0	No tunneling required
drone1	255.3 → 0.3 → 1.2
drone2	255.3 → 0.3 → 1.4 → 2.2
drone3	255.3 → 0.3 → 1.4 → 2.4 → 3.2
drone4	255.3 → 0.3 → 1.4 → 2.4 → 3.4 → 4.2
drone5	255.3 → 0.3 → 1.4 → 2.4 → 3.4 → 4.4 → 5.2

For example, the following procedure must be followed to connect to the controller Raspberry Pi of "drone1" from the GCS laptop. First, an SSH connection to `pi0_0` at IP address 192.168.255.3 must be established. From there, an SSH connection must be made to `pi0_1` at 192.168.0.3 because it has a connection to the "drone1" subnet through its wireless interface. Then, a final SSH connection can be made to `pi1_0` at IP address 192.168.1.2 on the "drone1" subnet. Alternatively, a single, well-built SSH command can be executed to tunnel to `pi1_0`. This command is shown below:

```
#Command to tunnel to drone1's controller Raspberry Pi
ssh 255.3 'ssh 0.3 'ssh 1.2''''''''
```

If public/private RSA key pairs are not shared between the GCS laptop and `pi0_0` and between each of the subsequent Raspberry Pis in the tunnel, then SSH requires passwords to be entered to establish the session. This password entry step cannot be automated; thus,

RSA authentication was configured for every pair of Raspberry Pis involved in automated SSH tunneling in the network.

4.3 Powering the Payload Devices

The M100 has several ports that are reserved for connecting and powering payload devices. These ports are shown in Figure 4.1. The maximum continuous output current for the XT30 and XT60 ports is 10.0 A, and the output voltage for each is between 20.0 V and 26.1 V.

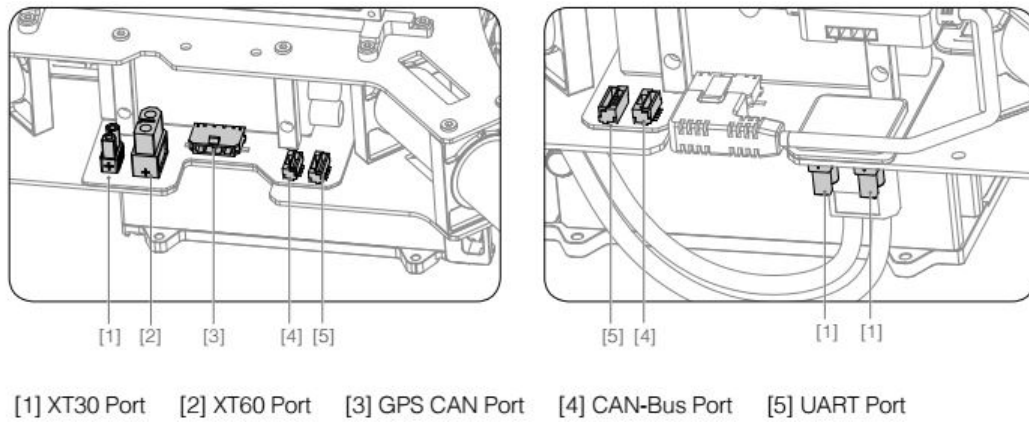


Figure 4.1. The Matrice 100's Reserved Ports. Source: [20].

The typical current consumption of a single Raspberry Pi 3 Model B is estimated to be 400 mA. The maximum total current draw of all four USB ports on the Raspberry Pi is 1.2 A, and the required voltage is 5.0 V. Likewise, each Linksys WRT54GL router requires 12.0 V and 1.0 A. In total, the five payload devices do not exceed the maximum output current for the Matrice 100's XT30 and XT60 ports.

The Raspberry Pis were powered with UBEC DC-DC step-down buck converters . Each of these provided 5.0 V and 3.0 A output. LM2596 DC-DC buck converters were used for powering the Linksys WRT54GL routers. These step-down converters were adjusted to output 12.0 V.

Input and output leads were soldered to both sides of the converters such that all of the payload devices could be powered by one XT30 port on their assigned M100. One of the six converter circuits is shown in Figure 4.2.

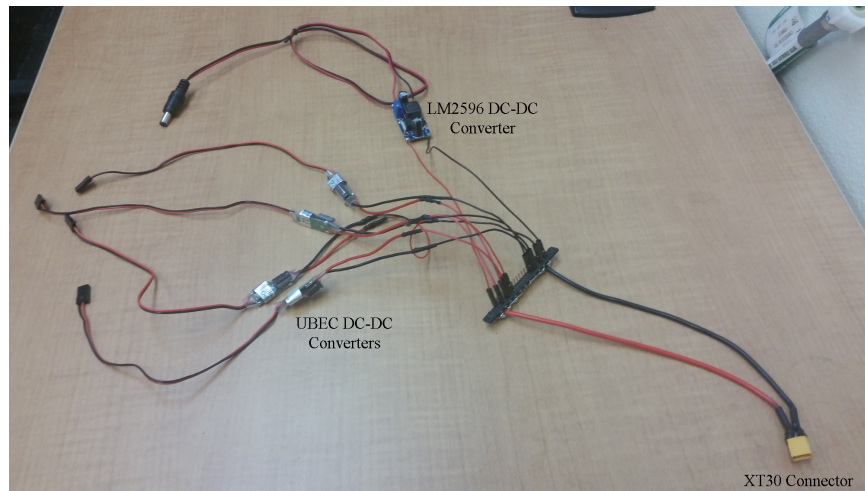


Figure 4.2. Converter Circuit Used to Power the Payload on a Matrice 100

4.4 Mounting the Payload

On each M100, the Raspberry Pis and router were mounted as shown in Figure 4.3. The Raspberry Pi that ran the onboard SDK was mounted to the top of the battery compartment. Then an expansion bay was mounted on top of the battery compartment, and the router was mounted to the top of that expansion bay with M2.5 hex screws. The remaining three Raspberry Pis were mounted on top of an additional expansion bay. The stacked Raspberry Pis were separated with M2.5 nylon spacers.

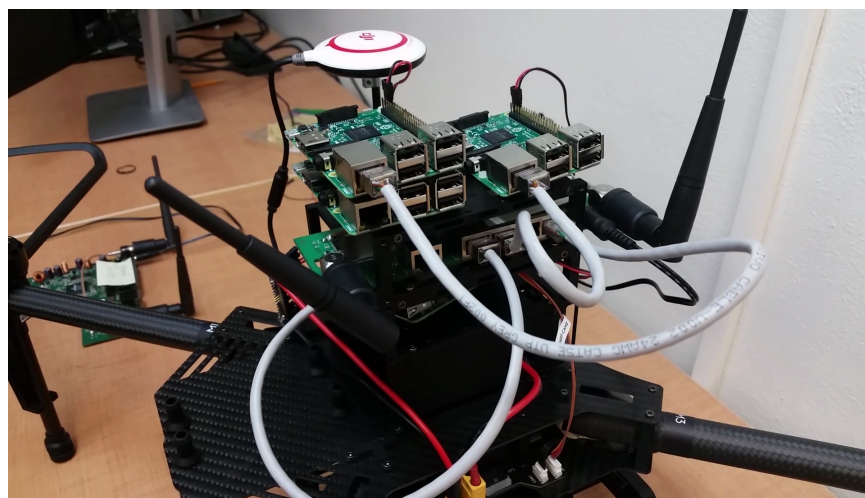


Figure 4.3. The Payload Mounted on a Matrice 100

CHAPTER 5:

Control and Communications Programming

The next step was to create a program to simultaneously control all six Matrice 100 quadcopters from a single laptop. This required creating both an onboard control program that would run on a Raspberry Pi and a graphical user interface (GUI) program that would run on a laptop. In the following sections, we describe how these programs were implemented.

5.1 Flight Controller Program

As mentioned in Chapter 2, the Matrice 100's remote controller uses a proprietary protocol to send flight control signals to the aircraft. In addition, DJI does not provide a way to control multiple aircraft from a single remote controller. For these reasons, an alternative method of flight control was implemented. At a minimum, the onboard flight controller program needed to provide functions for positional and GPS waypoint control. It also needed to implement an improved return-to-home (RTH) feature that caused the platform to retrace the previously executed commands. The default return-to-home feature, which causes the quadcopter to rise to a fixed altitude of 30.0 m and fly directly back to its initial takeoff position, was inadequate in locations with obstacles.

DJI's onboard SDK was used to implement the flight controller program. The programming workflow for a typical program written with DJI's onboard SDK is shown in Figure 5.1. First, an API key must be obtained from DJI. This key is used to activate the M100/Raspberry Pi pair. Next, the remote controller must transfer control of the aircraft to the Raspberry Pi. At this point, the remote controller regains control if the control program terminates or the Raspberry Pi fails.

5.1.1 DJI Onboard SDK Sample Program

DJI provides a sample program written in C++ that is compatible with Raspberry Pis and conforms to the programming workflow in Figure 5.1. This program provides a framework for developers to build upon when creating their own applications. The sample is an interactive program that runs from the command-line interface of the controller Raspberry

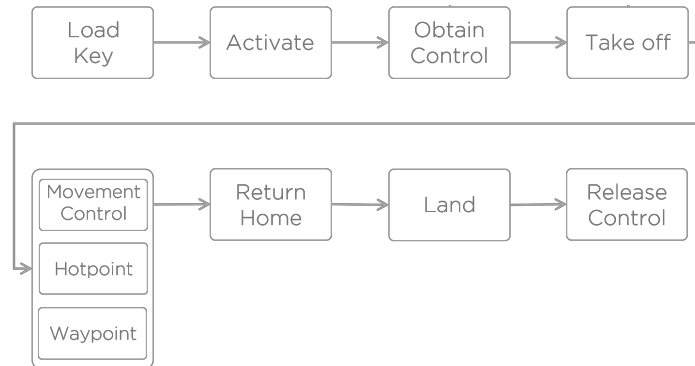


Figure 5.1. The DJI Onboard SDK Programming Workflow. Source: [28].

Pi. As shown in Figure 5.2, the program provides basic commands for obtaining control, taking off, and landing. The positional and waypoint control commands are limited in functionality, and the camera-related commands are irrelevant unless a supported camera is connected to the M100. To interact with this program and execute commands, the user must manually enter the character that corresponds to the command they wish to send to the quadcopter. For example, if the user wants to command the quadcopter to takeoff, he would enter the letter “e.” When executed, the commands are processed by the C++ program, sent across the serial link between the Raspberry Pi and M100, and processed by the flight controller. The flight controller then controls the M100 accordingly and responds to the Raspberry Pi with an acknowledgment. The sample program takes action according to the type of acknowledgment and then prompts the user to input a new command.

By default, to run the C++ sample, the following steps must be taken:

1. Turn on the M100 and select “P” mode on the RC.
2. Turn on the RC and wait for the M100’s LED to blink green.
3. Select “F” mode on the RC and execute the program.

Outside of a simulated environment, the user should not run this program while directly connected to the controller Raspberry Pi via keyboard and monitor because the UAV will attempt to fly and could create a safety hazard. For this reason, a wireless connection to the controller Raspberry Pi was created.

```

-----DJI Onboard SDK Interactive Sample-----
|
| The interactive mode allows you to execute a few commands
| to help you get a feel of the DJI Onboard SDK. Try them out!
|
| Standard DJI Onboard SDK Usage Flow:
| 1. Activate (The sample has already done this for you)
| 2. Obtain Control (The sample has already done this for you)
| 3. Takeoff
| 4. Execute Aircraft control (Movement control/Missions/Camera)
| 5. Return to Home/Land
| 6. Release Control (The sample will do this for you on exit)
|
| Available commands:
| [a] Request Control
| [b] Release Control
| [c] Arm the Drone
| [d] Disarm the Drone
| [e] Takeoff
| [f] Waypoint Sample
| [g] Position Control Sample: Draw Square
| [z] Local Mission Plan: Execute a pre-planned spiral
| [h] Landing
| [i] Go Home
| [j] Set Gimbal Angle
| [k] Set Gimbal Speed
| [l] Take Picture
| [m] Take Video
| [n] Exit this sample
|
| Type one of these letters and then press the enter key.
|
| If you're new here, try following the usage flow shown above.
|
| Visit developer.dji.com/onboard-sdk/documentation for more.
|
-----DJI Onboard SDK Interactive Sample-----

```

Figure 5.2. The Onboard SDK Sample Program. Source: [29].

5.1.2 Wirelessly Controlling the Matrice 100

The first objective in customizing the sample program was to determine how to control the program from a laptop on the same network as the controller Raspberry Pi. To do so, both the controller Raspberry Pi and the control laptop were connected to the same Linksys WRT54GL router. Then, an SSH connection was established between the GCS laptop and the controller Raspberry Pi. From there, the program was controlled as described in the previous section; however, this process became too cumbersome when a second drone was added to this scheme as controlling the quadcopters from multiple command-line interfaces was difficult and unsafe. As a result, a method for not only automating SSH connections but also intuitively controlling multiple UAVs that reside in different subnets was desired. A GUI control program was implemented in the Python programming language to meet this requirement. Before the GUI was implemented, a few problems arose.

5.1.3 Remote Controller Interference

When the second and third UAVs were added to the network, both the Linksys routers and the controller Raspberry Pis began to exhibit unstable behavior. The routers stopped leasing static IP addresses to the Raspberry Pis, and the SSH connections terminated prematurely. Remote controller interference was the source of the problem.

As mentioned in the programming workflow procedure, the M100 requires a constant connection with its remote controller, and this remote controller operates in the 2.4-GHz industrial, scientific, and medical radio (ISM) band. Likewise, the Linksys WRT54GL routers operate in the same frequency band. A spectrum analyzer was used to confirm this issue. The spectrum analyzer was configured to display frequencies between 2.39 and 2.5 GHz and to hold the maximum value of the received signals. First, a single Linksys WRT54GL router was turned on. The resulting trace is shown in Figure 5.3. Interference sources in the lab created the peaks on the left and right side of the display, and the Linksys router created the peak in the middle. The trace after the M100's remote controller was turned on is shown in Figure 5.4. Clearly, the remote controller was operating across the entire observed spectrum and at a higher power than the Linksys routers; therefore, it was necessary to determine a way to control the M100s without the remote controllers turned on.



Figure 5.3. Spectrum Analyzer Trace with Linksys WRT54GL Turned On

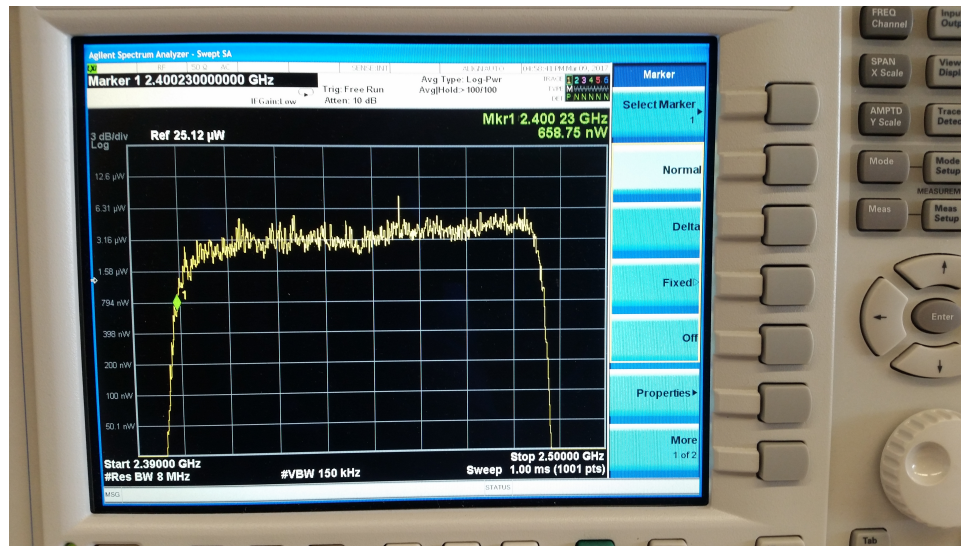


Figure 5.4. Spectrum Analyzer Trace with Matrice 100 Remote Controller Turned On

5.1.4 Virtual RC Protocol

The Virtual RC protocol was used to solve the remote controller interference problem. DJI created this protocol to provide all of the functionality of their physical remote controller from a programmatic perspective. The protocol creates a simulated remote controller for fine-grain flight control. In this thesis research, the protocol was used to convince the flight controller that a remote controller was connected to it and to transfer control to the controller Raspberry Pi. To do so, the protocol provides an “enable” command, which must be sent to the flight controller every 20.0 ms. With this workaround in place, the remainder of the C++ sample was modified to meet the requirements of this thesis.

5.1.5 Flight Controller Program Overview

As shown in Figure 5.5, the flight controller program starts by initializing the serial interface so that the Raspberry Pi and flight controller can transmit data to one another. Then, the interactive portion of the sample program is started. At this point, three threads are spawned by the parent thread. The first thread writes the quadcopter’s flight data to the standard output stream once every second. The second thread sends “enable” commands to the flight controller according to the Virtual RC protocol. The third thread reads user commands

from the standard input stream and forwards them to the final thread. The fourth thread uses a switch statement to process and execute the commands. After a timeout period or an exit command, the parent thread starts the reverse RTH routine and tells the other threads to terminate. The program ends once each thread has completed.

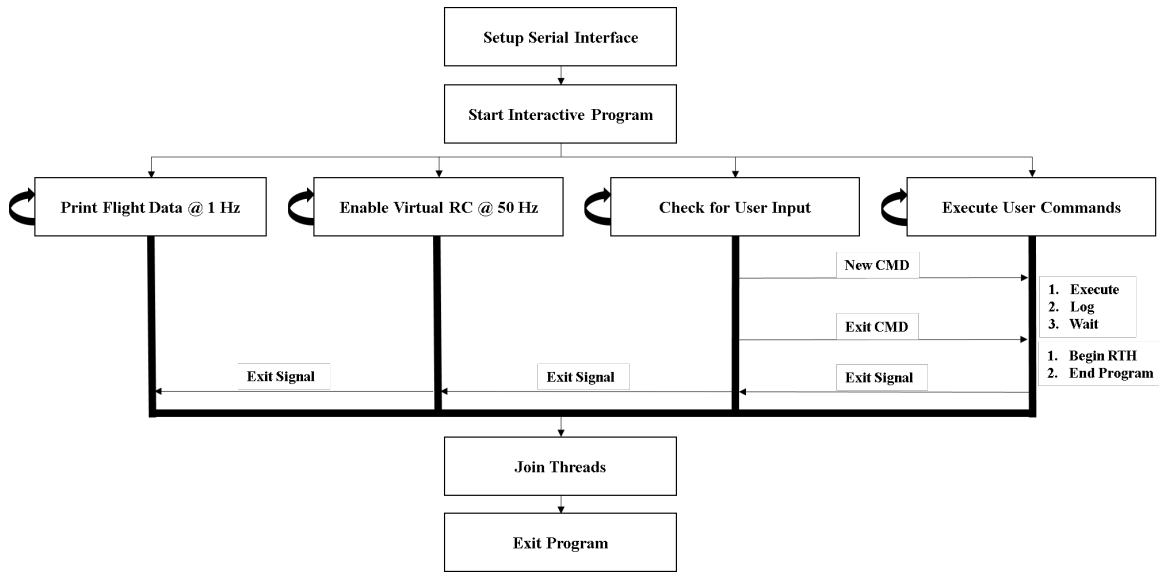


Figure 5.5. High-Level Flow of Control of the Flight Controller Program

5.1.6 Features of the Flight Controller Program

The following user commands were added to DJI's sample program:

- Positional commands for fine-grain, manual control of the aircraft
- GPS waypoint missions for autonomous flight through a given list of GPS coordinates
- Reverse RTH for recovery at the end of a flight or in case of signal loss
- Specific flight trajectories such as the lawnmower pattern mentioned in Chapter 1

The implementation of each of these features is briefly described in the following sections.

Position Control

The onboard SDK provides high-level functions for movement control. In the flight controller program, one of these functions was used to allow users to fly the aircraft in any direction by a given number of meters or change its heading by a given number of degrees.

For example, the following code listing shows how this function would be used if the user executed a command to fly the aircraft up by ‘z’ meters:

```
// Fly up by ‘z’ meters.  
int positionControlStatus = moveByPositionOffset(api, flight, 0, 0, z, 0);
```

Similar function calls were made for all other positional control commands.

Waypoint Control

The Matrice 100 is capable of autonomously flying through a list of GPS waypoints. The sample program provides an example function that causes the quadcopter to fly a square pattern based on a set of four relative GPS coordinates. In a similar fashion, a custom waypoint mission function was implemented for this thesis research. When the user executes a waypoint mission, he must include the number of waypoints and a list of the latitudes, longitudes, and altitudes for each waypoint. The flight controller program stores this information in a file. Then, the custom waypoint mission function uses the information in this file to create waypoint objects that are compatible with DJI’s high-level waypoint mission functions. Once each point is added to the mission, the `startWaypointMission()` function is called and the aircraft becomes fully autonomous until the completion of the mission.

Trajectory Control

Four advanced flight pattern functions were implemented: two for the lawnmower pattern and two for the perimeter pattern trajectories. The trajectory that was implemented in the `lawnmowerPatternUp()` function is shown on the left side of Figure 5.6. The function creates eight GPS waypoints based on the aircraft’s initial position when the command is executed. Each waypoint has an altitude of 100.0 m, and the overall area over which the trajectory is flown is 100.0 m × 100.0 m. The `lawnmowerPatternDown()` function assumes that the aircraft is located at the upper-right corner of the coverage area. The function implementation is identical to the `lawnmowerPatternUp()` function except that the flight pattern is rotated 180° about the origin. These two functions are intended to be used by two aircraft simultaneously to provide optimal coverage to ground units. Likewise, the perimeter pattern functions were implemented in the same manner.

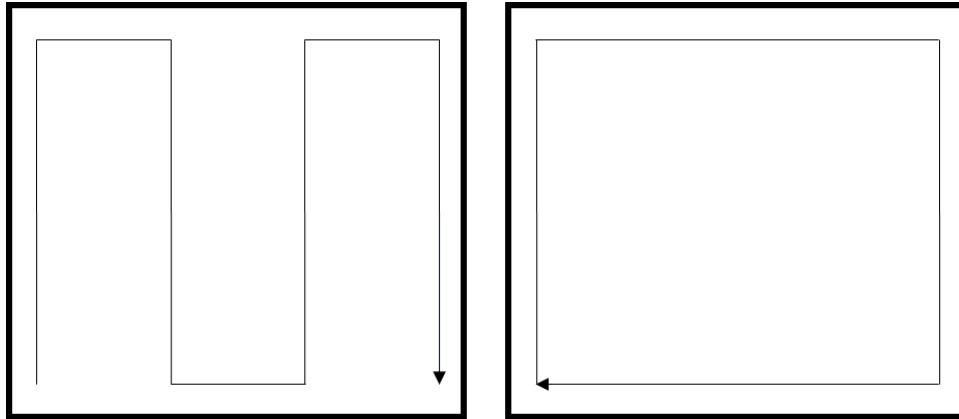


Figure 5.6. The “Lawnmower Up” (left) and “Perimeter Up” (right) Flight Patterns

Reverse Return-to-Home

The default failsafe behavior of the Matrice 100 is DJI’s return-to-home function. This return-to-home is triggered when the RC signal is lost for three or more seconds, the battery reaches a critically low level, or the user manually sends an RTH command. When one of these events occurs, the aircraft ascends to a user-specified altitude (30.0 m by default) and follows a straight-line path back to the home point, which is the GPS location recorded upon takeoff. In an environment with obstacles and heavy terrain, DJI’s default RTH function is insufficient and might lead to collisions; therefore, a better RTH was implemented.

The reverse RTH routine is triggered in three scenarios: the user executes a reverse RTH command, no commands are executed over a period of five minutes, or the user attempts to exit the program while the quadcopter is flying. In the normal operation of the flight controller program, the executed user commands and waypoint missions are stored in reverse order in dynamic queues; therefore, when a reverse RTH is triggered, the commands in these queues are iterated through to retrace the quadcopter’s flight path back to its initial position.

5.2 GUI Control Program

After the flight controller program was implemented, a GUI program that could be used to send commands and receive data from each controller Raspberry Pi was created. As shown in Figure 5.7, the GUI control program starts by connecting to each UAV’s controller

Raspberry Pi and starting their flight controller program. The user may specify any number of UAVs to connect to as long as each UAV is equipped with the proper networking equipment and each component follows the IP addressing scheme. Once the connections are made, the GUI continuously checks for user input and both reads and displays the flight data that each UAV prints to its standard output stream. Additionally, several buttons are provided to give the user more options for control and information gathering. The completed GUI is shown in Figure 5.8. The program ends when the user presses the “Quit” button.

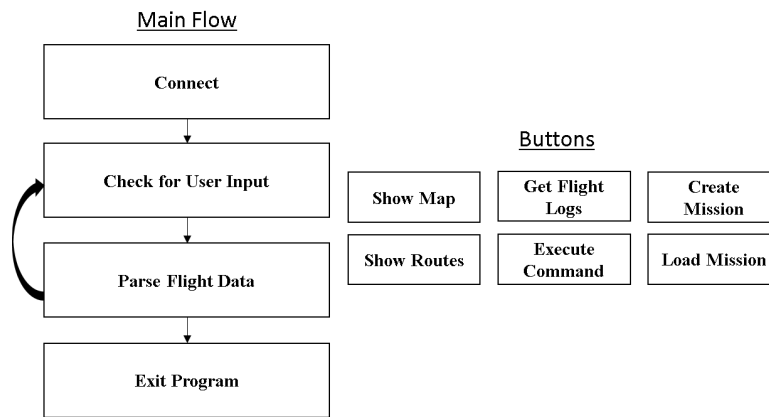


Figure 5.7. High-Level Flow of Control of the GUI Control Program

5.2.1 Features of the GUI Control Program

The implementation of the GUI program’s most important features is described in the following sections.

The “Connect” Button

As described in Chapters 3 and 4, transparent multi-hop SSH was used to connect from the laptop to the controller Raspberry Pis. The first time that the GUI’s “Connect” button is pressed, several events take place:

1. The command list for connecting to each UAV is created.
2. Python’s Paramiko library is used to attempt to connect to each UAV, using the created command list.

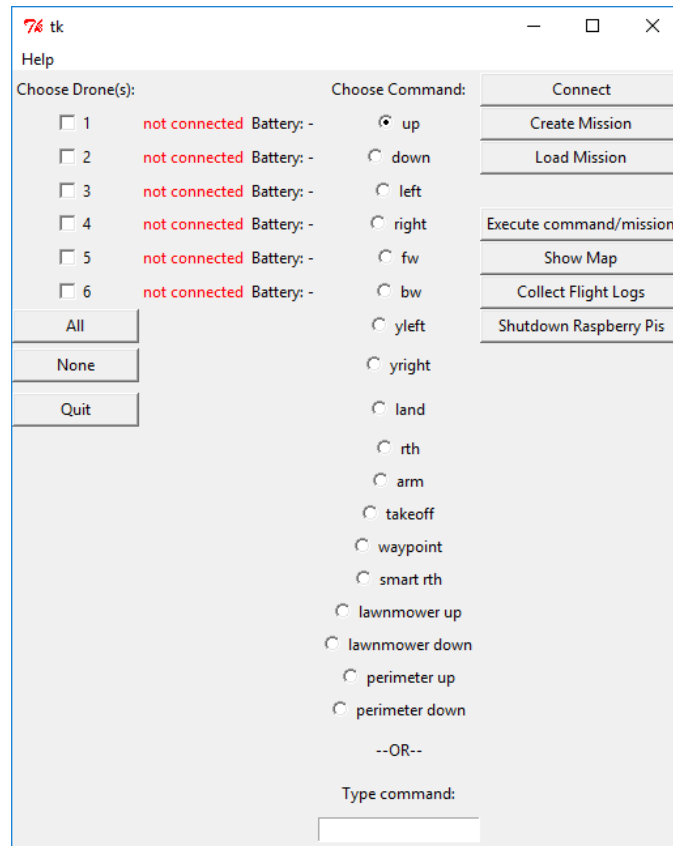


Figure 5.8. The GUI for Controlling a Network of Six UAVs

3. The flight controller program is started on each controller Raspberry Pi.
4. The interface is updated to display whether the connections were successful.
5. The GUI program begins parsing flight data and checking for user input.

If the “Connect” button is pressed again, a message box appears to notify the user that the connections have already been made. If some or all of the connections failed, then the user is prompted to check that the network is properly configured and restart the program.

Interfacing with the Flight Controller Program

The GUI program maintains access to each SSH connection’s standard input and standard output streams. To read a single line from one of these connections, the following line of code is used:

```
#Read a single line from SSH connection ii's standard output stream
line = ssh_stdout[ii].readline(2048)
```

Each UAV's controller Raspberry Pi continuously prints its battery percentage, velocity data, GPS location, altitude, and flight status to its standard output stream at a frequency of 1.0 Hz. This flight data is parsed by the GUI program and displayed to the user. An example of how this information is parsed is shown in the following listing:

```
#Parse the battery information and add to the GUI
if(line.startswith('Battery')):
    #Strip white space from both sides of string
    batteryPercentage = line.strip()
    #Add to UAV ii's flight data snapshot
    content[ii].append(batteryPercentage)
    #Update battery percentage label on GUI
    Label(master, text=batteryPercentage).grid(column=4, row=ii+1)
```

First, if the line starts with the string “Battery,” then the GUI program strips all leading and trailing white space characters from the line. The program then adds the resulting string to both an array containing the rest of UAV ii's flight data and the appropriate row and column of the GUI such that the user can observe the remaining battery percentage of this UAV.

On the other hand, when the user wishes to execute a command on a particular UAV, the GUI must write the command to the corresponding SSH connection's standard input stream. For example, to send a given command to UAV ii, the following lines of code are executed:

```
#Write command to SSH connection ii's stdin
ssh_stdin[ii].write(chosenCommand + '\n')
#Flush SSH connection ii's stdin buffer
ssh_stdin[ii].flush()
```

In this example, the chosen command is written to the SSH connection's standard input stream, and the newline character is equivalent to pressing the “Enter” key, which causes the flight controller program to read in the command. The flush() method ensures that the command is written to the standard input stream immediately rather than being stored in an internal buffer until later [30]. In this way, the GUI is able to transmit flight control commands to any of the UAVs.

The Waypoint Mission Buttons

The GUI provides buttons for both creating and executing waypoint missions. Waypoint missions must be uploaded from comma-separated values (CSV) files. As such, when the user presses the “Create Mission” button, the GUI program opens Litchi’s Mission Hub, which provides a simple web interface for creating these types of files [31]. The waypoint mission file can then be loaded into the GUI program with the “Load Mission” button. The GUI program iterates through the contents of this file and stores them in a list. Then, when the user executes a waypoint mission command, the contents of this list are iterated through and written to the appropriate SSH connection’s standard input stream. The flight controller program uses this information to execute the mission.

Collecting Flight Logs

For troubleshooting purposes, each UAV stores its flight data in two log files. The first log file contains a list of user-executed commands, flight status numbers, and the status of the timeout counter. The second log file contains the flight data that the UAV’s controller Raspberry Pi continuously prints to the standard output stream, including its GPS location. A Python script was created to connect to each UAV and collect these log files. The user can run this script by pressing the “Collect Flight Logs” button; the flight logs are also collected when the “Quit” button is pressed.

The collection script assumes that the UAVs are all within range of the ground router so that all of their controller Raspberry Pis are connected to it via their wireless interfaces. When the script begins, the laptop establishes SSH connections with each controller Raspberry Pi and copies the flight and GPS log files to its current working directory. Each file is named according to the UAV from which it was collected, and a timestamp is appended to the file name for uniqueness.

Mapping the UAVs

The GUI program also uses the flight logs to create maps of each UAV’s location. A Python module called “simplemap” was used to create these maps [32]. When the user presses the “Show Map” button, each UAV’s most recent GPS location is plotted in Google Maps. The markers for each UAV can be clicked to reveal the rest of that UAV’s instantaneous flight data.

Likewise, a script was created to display the flight path of a UAV in the network. Given the GPS log from a particular UAV, the “Show Routes” script makes use of the “simplemap” module to connect each of the GPS coordinates and display the route that was taken during that UAV’s flight.

Shutting Down the Raspberry Pis

Finally, the “Shutdown Raspberry Pis” button was created to provide the user with a convenient way to shutdown all 24 Raspberry Pis at the end of a test flight. When the button is pressed, the GUI program establishes SSH connections to each controller Raspberry Pi in the network. In turn, each controller Raspberry establishes SSH connections to the other three Raspberry Pis on its UAV and executes the shutdown command. Subsequently, the controller Raspberry Pis shut themselves down, and the GUI then displays a message box to notify the user whether the shutdowns were successful.

5.3 Communications Program

Next, a Voice over IP (VoIP) service was configured for the network. Software for setting up and tearing down VoIP sessions was implemented. Mumble was the VoIP service used for this project because it is open source and compatible with Raspberry Pis [33]; thus, Mumble servers were installed on one Raspberry Pi per UAV, which gave these Raspberry Pis the responsibility of acting as the VoIP server for their subnet. Any user with a Mumble client application installed on his device could connect to the Mumble server in his subnet. SSH tunneling through local port forwarding was required to allow clients on different subnets to connect to one another.

In this network, if multiple users within the same subnet wish to have a VoIP session, then they all simply connect to their local subnet’s VoIP server; however, when any of these users belongs to a different subnet, an agreement needs to be made on which server the users should connect to. One possible solution is to determine which user is connected to the highest subnet number and have everyone connect to that subnet’s VoIP server. Several other possible solutions exist, but for the sake of simplicity, this approach was taken. To illustrate this idea, an example in which several clients are connected to different subnets within the multi-UAV network is shown in Figure 5.9. In this scenario, the laptop with IP

address 192.168.5.51 is connected to the highest subnet number, 5, whereas the other clients are connected to subnets 0 and 2. The clients connected to subnets 0 and 2 must use SSH tunneling to connect to the VoIP server on drone5. On the other hand, the laptop on subnet 5 does not have to use SSH tunneling.

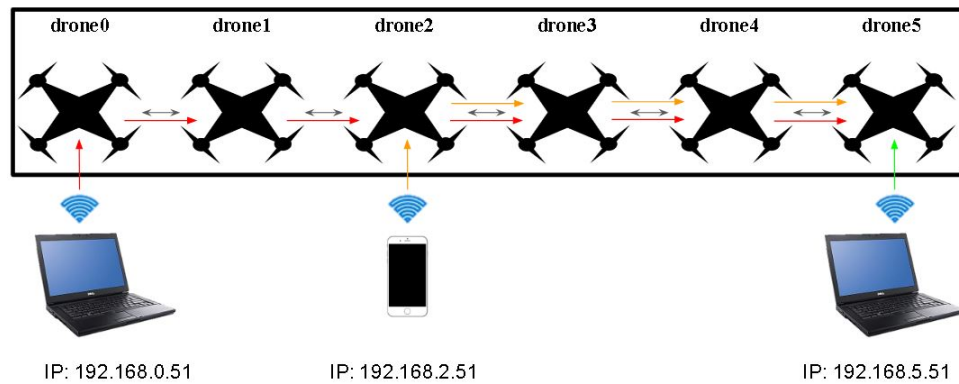


Figure 5.9. Example Usage of VoIP Service

The GUI that was created to automate the connection process for a two-client VoIP session is shown in Figure 5.10. To use the GUI, one client enters the IP address of the client to which he wants to connect. He then enters the username he wants for the current VoIP session and presses the “Go” button.

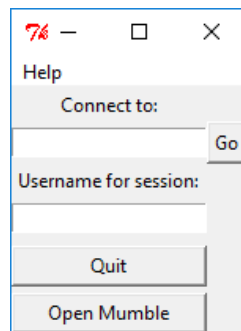


Figure 5.10. The GUI for Creating a VoIP Session

The underlying program then compares the source and destination clients' IP addresses to determine whether or not tunneling is necessary. If the source client's subnet number is greater than or equal to the destination client's subnet number, then the GUI notifies

the user to connect to the Mumble server on his own subnet. The program then inserts a new server entry into the client Mumble application's server table by editing the Mumble SQLite database. The server entry must include a server name, IP address, port, and client username. In this case, the program names the server 'LOCAL_SERVER' so that the user knows tunneling is not required. The program automatically fills in the Mumble server's IP address because each Mumble server was installed on the Raspberry Pis with IP addresses of the form 192.168.X.3 on each quadcopter's piX_1, where $X = 0, 1, 2, \dots, 5$ for a six-UAV network. Likewise, the port that each Mumble server listens on is 64738 by default, and the username is provided by the client.

If the source client's subnet number is less than the destination client's subnet number, the GUI notifies the user to connect to the Mumble server on the destination client's subnet. The program then generates the necessary SSH commands for local port forwarding and creates the tunnel between the client's Mumble application and the Mumble server. After the tunnel is created, the program updates the Mumble application's server table with a server named 'TUNNEL_TO_SERVER.' To connect to the Mumble server at 192.168.X.3 port 64738, the Mumble application sends its data via 127.0.0.1 port 2000 because an SSH tunnel is created and connected to the server via this IP address and port pair.

When the user presses the "Open Mumble" button, the Mumble client application is opened. The user can then connect to the "LOCAL_SERVER" or "TUNNEL_TO_SERVER" server, depending on which entry the program added to the application's server list. At the end of the VoIP session, the user must press the "Quit" button on the GUI, which causes the underlying program to establish SSH connections to each Raspberry Pi in the VoIP session's SSH tunnel and closes the ports that were used for local port forwarding.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:

Simulation, Testing, and Performance Evaluation

During the next phase, the flight control programs were tested in a simulated environment, indoor and outdoor flights were planned, and network throughput testing was conducted. These tests are described in this chapter.

6.1 Simulation Test Flights

The flight control programs were tested frequently during development. As a result, once the programs were modified to accommodate UAVs that conform to the IP addressing scheme, there were no major issues. In the indoor laboratory environment, all of the implemented features were tested. With all six M100s connected to the GUI program, takeoff and land commands were successfully executed on each quadcopter. These commands were sent to all six M100s simultaneously, to each pair of M100s sequentially, and also to each M100 sequentially. All of the combinations received the GUI's commands successfully.

Next, one of the quadcopters was connected via USB to the control laptop running the DJI Assistant 2 software. The software's flight simulator was used to ensure that the position, waypoint, flight trajectory, and reverse RTH functions were all implemented properly. All of the commands produced the desired behavior. Additionally, all of the flight data was parsed by the GUI program as intended, and the mapping functions worked as desired, as shown in Figure 6.1. In this case, drone1 was connected to the simulator, which initialized its GPS location to the position shown on the map.

6.2 Network Performance Evaluation

Next, the network's performance was tested within the indoor laboratory environment. Specifically, experiments were conducted to evaluate the quality of the VoIP service at various locations in the network. In addition, experiments involving file transfer and the use of a common network performance tool were conducted to determine the maximum achievable throughput of the network under various circumstances.

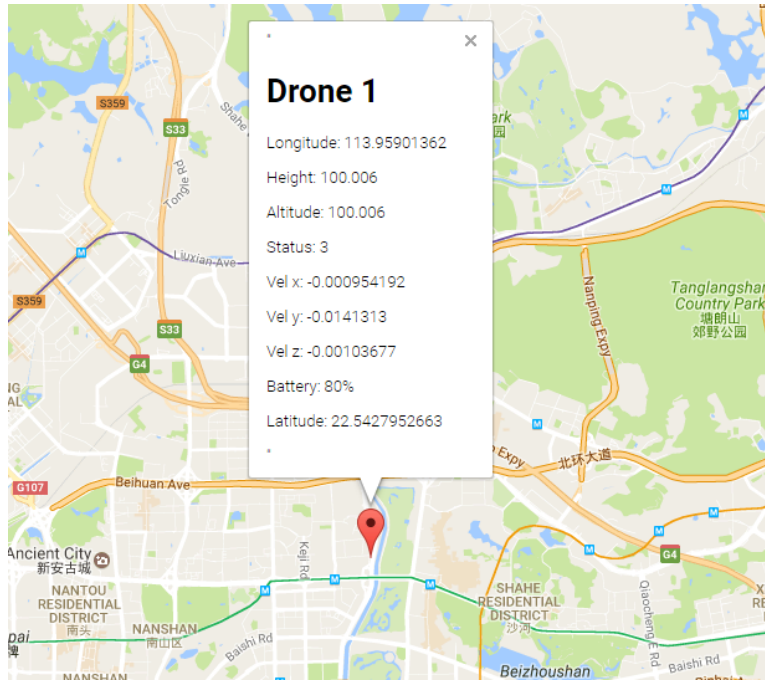


Figure 6.1. Successful Test of the GUI's "Show Map" Feature

6.2.1 VoIP Service Performance

First, the Mumble VoIP service was qualitatively evaluated. Two clients were connected to drone0's Mumble server, and both voice and text communications took place. The audio was loud and clear. Then, one client was moved to drone1's subnet so that the other client would have to tunnel to drone1's Mumble server, and the same qualitative tests were conducted. This process was repeated until one client was connected to drone5's subnet while the other still remained connected to drone0's subnet. At this point, the longest SSH tunnel was required to establish the VoIP session, yet the audio performance was indiscernible from that of the original session when no tunneling took place. This was likely because the Mumble service has low default bandwidth requirements (only 58.8 kbit/s) and excellent noise reduction [34]. Overall, it was concluded that regardless of the number of hops, the Mumble service provides acceptable audio and text communications for the network.

6.2.2 File Transfer and Throughput Testing

To quantitatively evaluate the network's performance, further benchmarking was required. First, simple file transfer experiments were completed. The first experiment involved two Windows laptops transferring a 50-MB file between one another at various locations in the network. Initially, both laptops were wirelessly connected to the "drone0" router. Then, while one laptop was left on the "drone0" subnet, the other laptop was connected to each of the remaining subnets. SSH tunnels were established between the client laptops to allow them to transfer the file between one another. The 50-MB file was generated with Windows' 'fsutil' command-line utility [35]. The file was transferred 15 times, and the recorded throughputs were averaged.

When both laptops were connected to drone0's router, the throughput was 5.20 Mbit/s. With one laptop connected to drone0's router and the other laptop connected to drone1's router, the throughput was 2.11 Mbit/s. The previous two results are shown in the Figure 6.2 as the "1 hop" and "2 hop" points, respectively. The hop count refers to the number of routers the data must travel through in order to reach the destination. Clearly, as the number of hops increased, the throughput decreased exponentially; however, even at six hops, the worst throughput was approximately 215 kbit/s, which is sufficient to support the Mumble VoIP service.

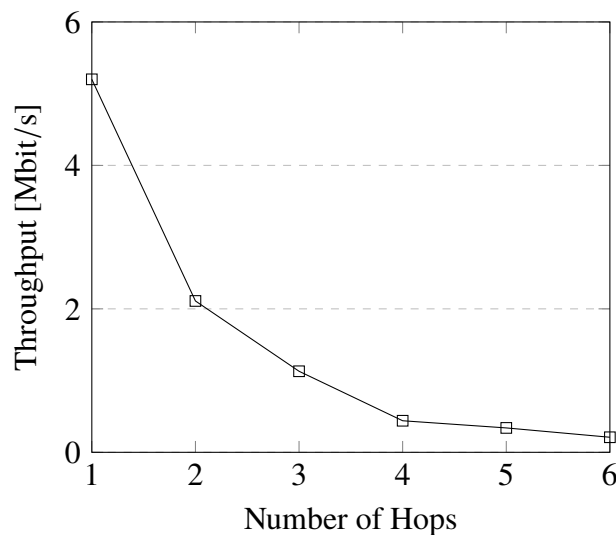


Figure 6.2. WinSCP 50-MB File Transfer Results

Nevertheless, the throughputs achieved in these initial tests were lower than expected. The Linksys WRT54GL routers support the IEEE 802.11g standard, which has a maximum theoretical throughput of 54.0 Mbit/s [36]. Realistically, the best expected throughput for a single wireless client connected to one of these routers is approximately 40 % to 50 % of the theoretical value, or 21.6 Mbit/s to 27.0 Mbit/s. With two wireless clients connected to the same wireless router, the expected throughput is once again cut in half to 10.8 Mbit/s to 13.5 Mbit/s because the data must travel from the first client to the router and then from the router to the second client. Still, the throughput for the single-hop scenario in this network was below the expected level. There are a number of potential reasons for this, including interference from the other routers and walls, packet loss, wireless channel contention, retransmissions, and the overhead of the WinSCP program.

The next experiment involved two Raspberry Pis running a common, open-source network benchmarking tool called iperf [37]. The tool was used for testing Transmission Control Protocol (TCP) throughput in the network and determining its maximum achievable bandwidth. One Raspberry Pi was connected to drone0's router and ran the iperf server, whereas the other Raspberry Pi was connected to each router in the network and acted as the iperf client. For each setup, the iperf throughput test was executed for five minutes, during which the program attempted to send as many TCP packets as possible. The results of this experiment are shown in Figure 6.3.

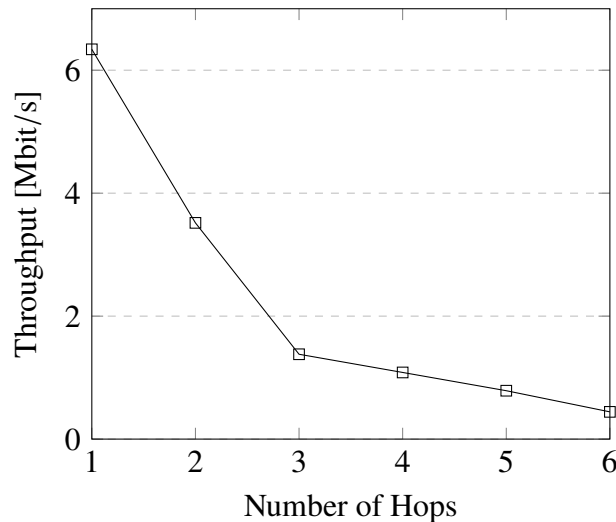


Figure 6.3. iperf Multi-Hop TCP Throughput Results

The single-hop throughput in this experiment was 6.34 Mbit/s, which was more than 1 Mbit/s larger than the single-hop throughput in the file transfer experiment. The remainder of the throughputs were slightly higher than the throughputs in the file transfer experiment as well. This likely indicates that the WinSCP program was contributing a significant amount of overhead because the none of the other conditions of the wireless channel were altered.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 7:

Conclusions and Recommendations

The idea of creating a communications relay network of UAVs to provide a link between ground and offshore units was presented in [3]. Given the conclusions of this work, the main purpose of this thesis research was to create a multi-UAV relay network that can be controlled by a single interface and extend the range of communications. With this network in place, the opportunities for further research are numerous. The results of this thesis research and recommendations for future researchers are presented in the following sections.

7.1 Summary of Work

One of the main accomplishments of this research was the implementation of a method for simultaneous flight control of six DJI M100 quadcopters. This required a carefully designed network architecture and a considerable amount of assembly and component integration. There are no known previously documented projects that allow for control of more than one DJI aircraft, which is most likely due to their main use as platforms for aerial imagery rather than research and development. Additionally, the flight control method was unique in its use of relayed commands in a chain topology, whereas relayed commands in a mesh topology or single-hop commands from centralized controllers are more common approaches. The network examined in this thesis research achieves extended ranges as a result of this design choice. Furthermore, networking applications for file transfer and voice and text communications were implemented and benchmarked. Overall, this network serves as a baseline for future variants in the Localization and Network Optimization Lab.

7.2 Future Work

The focus of future research should be on making the network adaptive and capable of handling failure. This will likely require a dynamic IP addressing scheme as opposed to the static IP addressing scheme that was defined for this thesis research. Additionally, an emphasis should be placed on increasing the autonomy of the network and flight control.

For example, the UAVs should be able to autonomously position themselves to optimize network performance. These programs should also be tested in the field to validate the control software and evaluate the network. The network performance should be analyzed under a variety of different conditions, such as with varying UAV and ground node mobility, increased range between access points, and with directional antennas equipped on the routers. Additional network performance metrics such as packet delay and jitter should be recorded to gain a better sense of the network's overall capabilities.

Finally, although the routers in this research were configured as access points, they could be reconfigured to act as mesh-networking nodes or repeaters as well. Experiments should be conducted with different network architectures and configurations so that they may be compared to one another.

List of References

- [1] Unmanned systems integrated roadmap, FY2013-2038. (2013). Department of Defense, Washington, DC. [Online]. Available: <https://www.defense.gov/Portals/1/Documents/pubs/DOD-USRM-2013.pdf>
- [2] P. G. Fahlstrom and T. J. Gleason, *Introduction to UAV Systems*, 2nd ed. West Sussex, UK: John Wiley and Sons, 2012.
- [3] B. Yanar, “Dynamic extension of network for cyber and communication,” M.S. thesis, Dept. of Elect. and Comput. Eng., Naval Postgraduate School, Monterey, CA, 2016.
- [4] S. Morgenthaler, T. Braun, Z. Zhao, T. Staub, and M. Anwander, “Uavnet: A mobile wireless mesh network using unmanned aerial vehicles,” in *IEEE Globecom Workshops*, 2012, pp. 1603–1608.
- [5] B. Li, Y. Jiang, J. Sun, L. Cai, and C.-Y. Wen, “Development and testing of a two-uav communication relay system,” *Sensors*, vol. 16, no. 10, 2016. Available: <http://www.mdpi.com/1424-8220/16/10/1696>
- [6] D. Hague, H. T. Kung, and B. Suter, “Field experimentation of cots-based uav networking,” in *IEEE Military Communications Conf.*, October 2006, pp. 1–7.
- [7] M. Asadpour, D. Giustiniano, K. A. Hummel, and S. Heimlicher, “Characterizing 802.11n aerial communication,” in *Proceedings of the 2nd ACM MobiHoc Workshop on Airborne Networks and Communications*. New York, NY: ACM, 2013, pp. 7–12. Available: <http://doi.acm.org.libproxy.nps.edu/10.1145/2491260.2491262>
- [8] E. Yanmaz, R. Kuschnig, and C. Bettstetter, “Achieving air-ground communications in 802.11 networks with three-dimensional aerial mobility,” in *Proc. IEEE INFOCOM*, April 2013, pp. 120–124.
- [9] E. Yanmaz, S. Hayat, J. Scherer, and C. Bettstetter, “Experimental performance analysis of two-hop aerial 802.11 networks,” in *IEEE Wireless Communications and Networking Conference*, April 2014, pp. 3118–3123.
- [10] Y. Gu, M. Zhou, S. Fu, and Y. Wan, “Airborne wifi networks through directional antennae: An experimental study,” in *IEEE Wireless Communications and Networking Conf.*, March 2015, pp. 1314–1319.

- [11] P. Zhan, K. Yu, and A. L. Swindlehurst, "Wireless relay communications with unmanned aerial vehicles: Performance and optimization," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 3, pp. 2068–2085, July 2011.
- [12] Buy Matrice 100 | DJI Store. (2017). [Online]. Available: <http://store.dji.com/product/matrice-100>. Accessed April 24, 2017.
- [13] Matrice 100 Specs. (2017). [Online]. Available: <https://www.dji.com/matrice100/info>. Accessed April 24, 2017.
- [14] Guidance. (2017). [Online]. Available: <http://wiki.dji.com/en/index.php/Guidance>. Accessed April 27, 2017.
- [15] Onboard SDK - Quick Start. (2017). [Online]. Available: <https://developer.dji.com/onboard-sdk/documentation/quick-start/index.html>. Accessed April 28, 2017.
- [16] WRT54GL Datasheet. (2017). [Online]. Available: <http://www.nerim.fr/docs/routeur-lynksys-WRT54GL.pdf>. Accessed April 29, 2017.
- [17] Raspberry Pi 3 Model B. (2016). [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Accessed April 29, 2017.
- [18] J. A. Guerrero and R. Lozano, *Flight Formation Control*, 1st ed. London: Wiley-ISTE, 2012.
- [19] ssh | BSD General Commands Manual. (1999). [Online]. Available: http://linuxcommand.org/man_pages/ssh1.html. Accessed April 29, 2017.
- [20] *DJI Matrice 100 User Manual*, DJI, Shenzhen, China, 2016, version 1.6.
- [21] DJI Tutorials. (2015, Aug. 23). DJI – Matrice 100 – Installation Demonstration. [YouTube video]. Available: <https://www.youtube.com/watch?v=HjSYI7YOgIs>. Accessed May 2, 2017.
- [22] DJI Support. (2016, Feb. 4). DJI Matrice 100 - Matrice 100 Unboxing and Setup. [YouTube video]. Available: https://www.youtube.com/watch?v=3Mji_y8uhI0. Accessed May 2, 2017.
- [23] DJI. (2015, Aug. 17). DJI Guidance Tutorial: Mounting the Guidance to the M100. [YouTube video]. Available: <https://www.youtube.com/watch?v=HjSYI7YOgIs>. Accessed May 2, 2017.
- [24] DJI, Shenzhen, China. (2017). *DJI Assistant 2v1.0.5*. [Online]. Available: <http://www.dji.com/matrice100/info#downloads>. Accessed May 12, 2017.

- [25] M. Thompson and P. Green. (2017). *Raspbian*. [Online]. Available: <https://www.raspbian.org/>. Accessed May 12, 2017.
- [26] T. Davis and J. Davis. (2017). *Win32 Disk Imager*. [Online]. Available: <https://www.raspbian.org/>. Accessed May 12, 2017.
- [27] Sebastian Gottschall / NewMedia-NET. (2009). *DD-WRT*. [Online]. Available: <http://www.dd-wrt.com/site/support/router-database/>. Accessed May 12, 2017.
- [28] Programming Guide - Using the Onboard SDK APIs. (2016). [Online]. Available: <https://developer.dji.com/onboard-sdk/documentation/application-development-guides/programming-guide.html>. Accessed May 8, 2017.
- [29] DJI Onboard SDK C++ Linux Example. (2016). [Online]. Available: <https://developer.dji.com/onboard-sdk/documentation/github-platform-docs/Linux/README.html>. Accessed May 8, 2017.
- [30] Python File flush() Method. (2017). [Online]. Available: https://www.tutorialspoint.com/python/file_flush.htm. Accessed May 14, 2017.
- [31] Mission Hub - Litchi. (2017). [Online]. Available: <https://flylitchi.com/hub>. Accessed May 14, 2017.
- [32] patrick-/simplemap. (2017). [Online]. Available: <https://github.com/patrick-/simplemap>. Accessed May 14, 2017.
- [33] Mumble Project. (2017). *Mumble VoIP*. [Online]. Available: <https://github.com/mumble-voip/mumble/blob/master/LICENSE>. Accessed May 12, 2017.
- [34] Mumble Wiki. (2016). [Online]. Available: <https://wiki.mumble.info/wiki/FAQ/English>. Accessed May 14, 2017.
- [35] Microsoft. (2017). *Fsutil*. [Online]. Available: <https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/fsutil.mspx?mfr=true>. Accessed May 12, 2017.
- [36] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11, 2007.
- [37] ESnet/Lawrence Berkeley National Laboratory, Berkeley, CA. (2017). *iperf*. [Online]. Available: <https://github.com/esnet/iperf>. Accessed May 12, 2017.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California